Fast and Accurate Traffic Measurement with Hierarchical Filtering

Haibo Wang, Student Member, IEEE, *Hongli Xu, Member, IEEE, Liusheng Huang, Member, IEEE, Yutong Zhai

Abstract—Sketches have been widely used to record traffic statistics using sub-linear space data structure. Most sketches focus on the traffic estimation of elephant flows (i.e., heavy hitters) due to their importance to many network optimization tasks, e.g., traffic engineering and load balancing. In fact, the information of aggregate mice flows (e.g., all the mice flows with the same source IP) is also crucial to many security-associated tasks, e.g., DDoS detection and network scan detection. However, the previous solutions, e.g., measuring each individual flow or using multiple sketches for independent measurement tasks, will result in worse estimation error or higher computational overhead. To conquer the above disadvantages, we propose an accurate traffic measurement framework with multiple filters, called Sketchtree, to efficiently measure both elephant flows and aggregate mice flows. These filters in Sketchtree are organized in a hierarchical manner, and help to alleviate the hash collision and improve the measurement accuracy, as the number of flows through hierarchical filters in turn will be decreased gradually. We also design some mechanisms to improve the resource utilization efficiency. To validate our proposal, we have implemented Sketchtree and conducted experimental evaluation using real campus traffic traces. The experimental results show that Sketchtree can increase the processing speed by 100%, and reduce the measurement error by over 30% compared with state-of-the-art sketches.

Index Terms—Network Measurement; Heavy Hitters; Hierarchical Filtering; Sketch; Attribute.

1 INTRODUCTION

NETWORK measurement, with its goal to estimate the traffic size of network flows, is crucial to helping network operators make better network management decisions. To pursue better measurement accuracy, different solutions, e.g., using flow tables and packet sampling, have been proposed. However, due to the limited resources (e.g., TCAM and CPU computing capacity) on switches, these solutions often face with some disadvantages. For example, SDN switches can measure the traffic size using flow tables (or flow entries). Since the number of flows (e.g., $10^6$) usually far exceeds the flow table size (e.g., 16K on most switches [1]), it is impossible to measure the traffic size of each individual flow only using the flow tables. Meanwhile, packet sampling measures flow information with a predefined sampling rate, e.g., 0.01. However, it suffers from low measurement accuracy as only partial flows are sampled and some mice flows may be discarded. Although the measurement accuracy can be improved by increasing the sampling rate or even recording all the traffic, e.g., SPAN [2], the required resources, including memory and computing, will dramatically increase and pose scalability issues especially in high-speed networks.

- H. Xu (Corresponding Author), L. Huang and Y. Zhai are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230027, and also with Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu, China, 215123. E-mail: xuhongli@ustc.edu.cn, lshuang@ustc.edu.cn, zyt1996@mail.ustc.edu.cn.
- H. Wang is with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230027, and also with the Department of Computer and Information Science and Technology, University of Florida, Gainesville, FL, USA, 32611. E-mail: wanghaibo@ufl.edu.

Sketches provide an alternative solution to achieve fine-grained traffic measurement with compact data structures, which summarize traffic statistics of all packets with fixed-size memory. Due to resource constraints on switches, most sketches focus on the statistics of elephant flows, as these flows are usually relevant in many network optimization tasks. For instance, heavy hitters identify large flows whose byte volumes are above a threshold [3], and load balancing can be achieved by redirecting some elephant flows [4].

Aside from elephant flows, some statistics information derived from mice flows is of equal importance for many tasks. In particular, some security-associated tasks require the traffic information of aggregate mice flows with different attributes (e.g., all the mice flows with the same source IP prefix) [5], regardless of the information of each individual mice flow. For example, DDoS attack refers that a number of sources send packets to a restricted destination [6]. To detect this attack, we need the overall traffic amount of (mice) flows to a destination. For convenience, we call these mice flows are aggregated in the attribute of destination address. Another example is network scan detection [7] [8] [9], which requires the overall traffic of mice flows from a specific source (i.e., aggregate mice flows in the attribute of source address).

Therefore, traffic measurements of both elephant flows and aggregate mice flows are necessary. One may think that it is easy to infer the statistics of aggregate mice flows with the information of each individual (mice) flow. However, this solution poses two challenges. (1) Though many sketches work well for elephant flows, they cannot guarantee high measurement accuracy for individual mice flows. For instance, our experimental results on some sketches (e.g., Count-Min [10], Cold Filter [11]) show that the average
measurement error of mice flows is several times of their real frequency. Especially, if a mice flow is misreported as an elephant flow, the estimation error will be much larger. Thus, the sum of all individual flows in an aggregate mice flow may violate its real frequency a lot, which will be validated through simulations in Section 6.4. (2) Sometimes it is difficult to obtain all the individual flows in an aggregate mice flow. For example, to estimate an aggregate mice flow towards a certain destination in the DDoS attack, we need to find out all the sources in the network, which is time and resource consuming.

Another way for traffic measurement of both elephant flows and aggregate mice flows is adopting one independent sketch for each specific measurement task. For example, to detect heavy hitters and DDoS, we need the information of elephant flows and aggregate mice flows in the attribute of destination address. Elephant flows can be recorded by Count-Min, and an additional Count-Min sketch needs to be applied to record the traffic information of aggregate mice flows in the attribute of destination address. Note that these two Count-Min sketches are mutually independent, as they hash different parts of the packet header into distinct data structures. Moreover, if there are multiple security-associated tasks, e.g., detection of DDoS and network scan, the aggregate mice flows ought to be recorded in different attributes. As a result, multiple independent sketches are required. This solution requires each sketch to independently process all the packets in the data stream, which is unaffordable for switches as their computational resources are limited, especially with multiple measurement tasks.

To this end, we propose an accurate traffic measurement framework for both elephant flows and aggregate mice flows. This framework comprises multiple filters, each associated with a specific task. Each filter will send the task-related flows to the corresponding sketch and others to the next filter for further measurement. Since this multi-filter framework looks like a binary tree, we call it Sketchtree, which is task-oriented and adaptive to different tasks. Specifically, aggregate mice flows are measured in different attributes, which can be dynamically configured according to the different requirements of tasks. Our main contributions can be summarized as follows:

- We have formally defined the multi-attribute heavy hitters (MTHH) problem, whose objective is to find out all the heavy hitters in a given set of attributes, including the traditional elephant flows and the aggregate mice flows in certain attributes whose traffic size exceeds a predefined threshold. We have presented the difference between MTHH and two related problems, heavy hitters (HH) and hierarchical heavy hitters (HHH).
- We inherit the idea of filtering, and apply multiple filters into Sketchtree. Specifically, Sketchtree uses the first filter to separate elephant and mice flows, and other filters to separate aggregate mice flows in different attributes. These filters in Sketchtree are organized with a hierarchical manner, which helps to gradually decrease the number of flows through hierarchical filters in turn, and further reduces the probability of hash collision in traffic measurement. Thus, Sketchtree can achieve high measurement accuracy of heavy hitters in different attributes.
- To reduce the memory and computation consumption, we design some optimization methods, such as building Huffman tree [12] by adjusting the orders of filters that process aggregate mice flows in different attributes. With these methods, our solution can achieve computation and memory efficiency.
- We have implemented Sketchtree. The experimental results on real traffic traces show that Sketchtree can largely improve the measurement accuracy of heavy hitters in different attributes and the processing speed compared with state-of-the-art methods like Count-Min [10] and Cold Filter [11], under the same memory size.

The rest of the paper is organized as follows. Section 2 introduces background of traffic measurement, defines the multi-attribute heavy hitters (MTHH) problem and proposes two baseline solutions. In Section 3, we present overview of our proposed solution, i.e., Sketchtree. Section 4 gives algorithms description and two optimization methods of Sketchtree. The theoretical analysis of Sketchtree is presented in Section 5 and Section 6 gives the performance evaluation. Finally, we conclude this paper in Section 7.

## 2 Background and Motivation

This section first introduces some typical sketches, and then presents challenges related to network measurement.

### 2.1 Related Works and Typical Sketches

Network measurement has been extensively studied in the context of different techniques such as sampling [13] [14], wavelets [16] [17], and sketches [10] [18] [19] [20] [21] [22] [23] [24]. Among these techniques, sketches are widely used for its high processing speed, low memory consumption and high measurement accuracy for (elephant) flows. The most acknowledged sketch is Count-Min (CM) [10], which consists of $d$ arrays, each associated with a hash function $h_i(\cdot)_{1 \leq i \leq d}$. For each incoming flow $f$ with frequency $N_f$, Count-Min hashes flow $f$ to $d$ counters $h_i(f)_{1 \leq i \leq d}$ using $d$ hash functions, and increments the values of all the $d$ hashed counters $value(h_i(f))$ by $N_f$. When querying the estimated frequency of this flow, Count-Min reports the minimum value among these $d$ hashed counters, i.e., $\min_{1 \leq i \leq d}{value(h_i(f))}$. Count-Min reports non-negative results for each flow compared with the real frequency. To reduce the estimated error, another sketch, called CM-CU [25], only increases the frequency of counter(s) whose values are smallest among $d$ hashed counters.

Some sketches, based on CM and CM-CU, support top-$k$ queries with additional memory, e.g., a heap [26], a hierarchical data structure [27], or an array with $k$ counters [18] [22] [28]. For instance, HeavyGuadian [23] uses an bucket table, each containing a heavy part and light part, to separate and guard the information of elephant flows, and Pyramid sketch [19] uses different layers of counters to automatically enlarge the estimation range of the corresponding counters according to the current frequency of the incoming
packet. These sketches can achieve high processing speed when the number of elephant flows is small, e.g., 32 in ASSketch [18]. If the number of elephant flows gets larger, the processing overhead may be unaffordable. Moreover, these sketches cannot measure the aggregate mice flows at the same time. The literatures about sketches for specifically recording aggregate mice flows are rare but the idea about estimating the aggregate flows is not new. For instance, individual flows can be aggregated by a certain matching rule (also called attribute), e.g., source or destination address, and then recorded by current counter-based sketches and top-\(k\) sketches [22] [29]. However, measuring the aggregate mice flows in one attribute needs one independent sketch to process the whole data stream. Therefore, multiple sketches have to be used to process the data stream independently to record the aggregate mice flows in different attributes, which may be unaffordable for current switches. The survey of more sketches can be found in [30] [31].

Instead of processing data stream in one sketch, recent research called Cold Filter [11] applies a filter to separate high-frequency flows from data stream at the first stage and then sends these high-frequency flows to the second stage ( aforementioned common sketches) for further measurement. Specifically, Cold Filter uses a two-layer counter-based sketch with small-size counters to record the frequencies of mice flows. If all the hashed counters overflow at both layers, Cold Filter will report the incoming flow as an elephant flow and send it to the existing sketches, e.g., CM-CU [25] and Space-Saving [28] for further measurement. Thus, Cold Filter needs to be combined with an existing sketch, and the combined framework excels at measuring heavy hitters.

2.2 Information of Elephant Flows and Aggregate Mice Flows Matters

To maintain qualified network performance, operators need to serve many tasks, e.g., load balancing and cyber security. These tasks require traffic statistics of heavy hitter in different attributes. Most of the previous works pay great attention to heavy hitters in one attribute, such as top-\(k\) heavy hitters. However, heavy hitters in multiple attributes are sometimes of equal importance in some security-associated tasks, e.g., DDoS detection. Here, we list some tasks that require the statistics of heavy hitters in different attributes as follows:

- **Top-\(k\) heavy hitters** [3]: identify top-\(k\) elephant flows during a time window.
- **Traffic changes detection** [41]: indicate flows which trigger the most traffic changes over two consecutive time windows. This task requires the information of elephant flows.
- **DDoS detection** [43] [44]: in DDoS attack, multiple hosts send more than a threshold of data to a specific destination host within a time window. DDoS attack can be detected using the traffic size of heavy hitters in the attribute of destination address.
- **Network scan detection** [8] [9]: during network scan, a source host sends more than a threshold of data to multiple destination hosts within a time window. To detect network scan, we need the traffic size of heavy hitters in the attribute of source address.

The above tasks require accurate information of heavy hitters in different attributes. For example, heavy hitter detection needs the traffic information of elephant flows whose byte volumes are above a predefined threshold, e.g., 1% of the link capacity during a time window. Network scan detection requires traffic information of heavy hitters in the attribute of source address. That is, it requires the overall traffic of mice flows from one source address, rather than the detailed traffic size of each individual mice flow. It is worth noting that the aggregation way of mice flows depends on the requirements of tasks. For a different task like DDoS detection, the traffic size of aggregate mice flows classified by the destination address is necessary. Therefore, we need to identify heavy hitters in different attributes accurately, which is the main motivation of this paper.

2.3 Heavy Computational Overhead

Current commodity switches are often equipped with limited computational capacity, which constrains the processing speed for different sketches. Let’s take the Count-Min sketch as an example. The inserting speed for Count-Min is about 10M packets per second [11]. For TCP/IP protocol, its packet size is at most 1500B. To be more practical, the average packet size is usually much less than the maximum packet size, e.g., ¼. That is, the throughput of the Count-Min sketch can be 10M×500B = 40Gbps. Since modern data center network scales to 40 Gbps or even higher speed [45], the throughput of lightweight sketches like Count-Min is qualified but still in the same magnitude of the bandwidth. However, CPU resources will become scarcer under following circumstances. (1) Some individual sketches with additional functions improve the measurement ability/accuracy, but also consume more CPU resources. For example, FlowRadar [36] can detect heavy changes by comparing the frequency of a flow in two adjacent time windows with an Invertible Bloom Lookup Table (IBLT) [36]. However, its computational overhead and throughput are 2584 CPU cycles per packet and 2.5Gbps (only ¼ of the link bandwidth 40Gbps) [22], respectively. Computational overheads of some existing sketches are shown in Table 1\(^2\). (2) Multiple sketches may be deployed on a switch to support different tasks. Although some lightweight sketches, like CM and CM-CU, are able to handle the incoming packets in real time, the switch becomes heavy-loaded if running multiple sketches simultaneously. Thus, we expect that multiple tasks, e.g., DDoS detection, network scan detection, are conducted by one “big” but lightweight sketch, rather than multiple individual sketches with each sketch handling one task.

2.4 Problem Specification

In this section, we formally define our problem as multi-attribute heavy hitters problem, which identifies all the

---

1. Due to the diversity of attacking methods, e.g., ICMP LAND, IP, etc. [42], the DDoS detection can be conducted through traffic volume (also called the number of packets) or the count of source hosts. This paper adopts the traffic volume of aggregate mice flows like [7], [42].

2. The computational overheads of MRAC, FlowRadar, RevSketch, UnivMon and Deltoid are cited from Sketchvisor [22], and the results of remaining sketches come from our experiments. The source code are publicized at Github [46].
elephant flows (without aggregation) and significant flows in different levels (attributes) of aggregation. Prior to that, we first review the problem definitions of heavy hitters and hierarchical heavy hitters [47] as follows:

**Definition 1.** [Heavy Hitters (HH)] Given a data stream with the total traffic amount $\Omega$ and a threshold $\phi(0 \leq \phi \leq 1)$. Heavy hitter is to find all the flows whose traffic size exceeds the volume $\phi \cdot \Omega$ and their associated flow size. More precisely, define that $s(f)$ is the traffic size of flow $f$ and the total traffic amount $\Omega = \sum_{f \in \Gamma} s(f)$ where $\Gamma$ is set of all flows in the data stream. The set of heavy hitters is $\{f | s(f) \geq \phi \cdot \Omega \}$.

**Definition 2.** [Hierarchical Heavy Hitters (HHH)] For any attribute $p$ in the hierarchical domain, we define $\Gamma(p)$ as the set of flows matching attribute $p$. Note that $\Gamma(p)$ does not include the heavy hitters in attributes that are descendants of $p$. $v(p)$ denotes the total traffic amount of all flows in $\Gamma(p)$, i.e., $v(p) = \sum_{f \in \Gamma(p)} s(f)$. The hierarchical heavy hitters problem is to find HH in all attributes. Formally, HHH outputs all the pairs, each pair with an HH and its corresponding attribute, i.e., $\{(v(p), p) | v(p) \geq \phi \cdot \Omega \forall p\}$.

Admittedly, the definition of a flow varies and for a certain attribute, HH can record flows in that attribute, which is part of the HHH’s job. However, the HHH problem will explore all the possibilities of attributes, which is the exponential function of the height of the domain hierarchy. Let destination IP as an example, its mask types can be up to $2^{32}$. Therefore, HHH is costly and may be burdensome for current hardware devices like switches.

To this end, this paper will find out the heavy hitters in the given set of attributes, which are limited in number and can be adjusted under different tasks. We define the multi-attribute heavy hitters problem as follows.

**Definition 3.** [Multi-Attribute Heavy Hitters (MTHH)] For any attribute $p$ in a given set of attributes $P$, MTHH finds out the heavy hitters in attribute $p$ whose overall traffic size exceeds the threshold, excluding all traffic from HH in attributes that are descendants of $p$. The MTHH problem will output the traffic size of these heavy hitters, i.e., $\{(v(p), v(D, p)) \geq \phi \cdot \Omega \forall p \in P\}$.

We explain the descendant of an attribute through an example. For instance, $p_1$ represents an attribute of source-destination address and $p_2$ represents another attribute of source address. We call $p_1$ the descendants of $p_2$. The selection of the attributes is determined by practical applications. For example, DDoS detection may need to know the heavy hitters in attribute of destination IP while network scan detection focuses on HH with attribute of source IP. It’s worth noting that the original heavy hitters without any attribute are to identify the individual elephant flows, which is an important topic. Therefore, in this paper, we consider the MTHH problem with $t + 1$ attributes, namely, $p_0, p_1, p_2, \ldots, p_t$, in which the attribute $p_0$ is the heavy hitters without flow aggregation. Accordingly, the heavy hitters in attribute $p_i$ are denoted by $HH-i$ flows. The objective of the MTHH problem is to find all the HH-i flows with $0 \leq i \leq t$.

### 2.5 Baseline Solutions

In this section, we introduce two baseline solutions from current literature that solves the MTHH problem. Since the hash operation accounts for the main computing cost of sketch-based solutions [48], we will present the hash times of these two solutions for comparison in terms of computing cost.

The first baseline solution applies one independent sketch (e.g., Count-Min or CM-CU) for each attribute $p$. As shown in Fig. 1, if there exist $t + 1$ types of attributes, $t + 1$ sketches will be adopted. Each sketch estimates the value of flows aggregated in the attribute $p \in P$. We can use the counter’s value to infer the traffic amount of all the flows excluding HH in attributes that are descendants of $p$. Therefore, the estimated value in sketch $i (1 \leq i \leq t)$ should subtract the volume of HH in attributes that are descendants of $p$. The subtraction is pretty tough as we have to store all the key of elephant flows in sketch 0. So the baseline solutions will not subtract value of these HHs.

The second baseline solution aims to solve the hash cost for each sketch. As shown in Fig. 2, it employs the filter to select the elephant flows whose estimated size is above a threshold $T$. Only these elephant flows will be measured carefully in the second sketch stage. Note that the number of packets in elephant flows takes up $\beta (\beta < 1)$ of the overall number of packets in the data stream and the numbers of hash functions used in the filter and the sketch are $d_1$ and $d_2$, respectively. As a result, the average hash times per packet is $(d_1 + d_2 \cdot \beta) \cdot (t + 1)$. According to [11], to achieve the same accuracy of elephant flows, the number of hash
function used in the second baseline solution is less than that in the first baseline solution, i.e., \(d_1 + d_2 < d\). Therefore, the computing overhead in the second baseline solution is reduced compared with the first baseline solution.

The above two baseline solutions require averagely \(D \cdot (t + 1)\) hash times per packet, where \(D\) is the hash times for a single sketch and \(t + 1\) is number of sketches that the data stream traverses. However, as we have mentioned in Section 2.3, the current switches struggle dealing with multiple independent sketches, especially when the number of sketches \(t + 1\) is large. Different from previous articles that consider reducing the hash times per packet, i.e., \(D\), this paper, on the contrary, tries to process the data stream once in one "big" sketch. Therefore, the hash times per packet of our solution will be \(\bar{D}\), where \(\bar{D}\) will be given in the first paragraph of Section 5.

### 3 Sketchtree Overview

Sketchtree is a robust network measurement framework for finding heavy hitter with different attributes. Assume that there are \(t\) attributes. Specifically, \(t\) also may be the number of security-associated tasks, e.g., 2 (one is the attribute of source address for network scan detection, the other is the attribute of destination address for DDoS detection), and is usually limited.

We observe that these different applications/tasks, including elephant-flow measurement and \(t\) security-associated tasks, usually require the information of mutually exclusive flows. On one hand, elephant-flow measurement only cares for those elephant flows, while security-associated tasks require the information derived from mice flows. On the other hand, for each security-associated task, it needs the information of heavy hitter in certain attribute \(p_i\), because only these flows can be potential attacks/dangers. Among different security-associated tasks, heavy hitters in different attributes usually don’t share individual mice flows, as a mice flow can hardly serve multiple security-associated tasks simultaneously [5] [42] [49] [50]. For example, a DDoS-associated flow may be sent to the destination host/server during Three-way Handshaking in TCP protocol, while network scan may be conducted after the completion of Three-way Handshaking. Thus, under most circumstances, these tasks measure different subsets of flows. Moreover, even if a mice flow serves multiple security-associated tasks, it is possible that the detection results of these tasks are not affected. That’s because the overall estimated frequency of heavy hitters in an attribute still exceeds the threshold, despite that one or several individual mice flows are missed. Thus, these different tasks care for different subsets of flows. In this paper, we propose to gradually separate these subsets of flows using different filters, so that each subset of filtered flows will be measured by different parts in Sketchtree. As a result, the whole framework works like a “big” sketch.

Fig 3 presents the architecture of Sketchtree, which consists of two stages: filtering stage and measurement stage. Generally, filtering stage includes \(t + 1\) filters, namely, Filters 0 to \(t\), and measurement stage consists of \(t + 2\) sketches, namely, Sketches 0 to \((t + 1)\). Each filter is responsible for flow separation and each sketch is responsible for flow measurement. Specifically, Filter \(i\) \((0 \leq i \leq t - 1)\) will send HH-\(i\) flows to Sketch \(i\) and remaining mice flows to Filter \(i + 1\). The only exception is the last filter, namely, Filter \(t\). It sends the remaining mice flows to Sketch \(t + 1\) for further measurement (Sketch \(t + 1\) is shown to preserve the consistency of the data processing of Sketchtree. It can be removed if the information of remaining mice flows is not required, like in the MTHH problem).

- **Elephant flows in Sketch 0**: all elephant flows (also called heavy hitter in attribute 0) will first be processed by Filter 0 and then sent to Sketch 0 for further measurement. Thus, all the information of elephant flows is stored in Sketch 0.
- **Heavy hitters in Sketch 1 to \(t\)**: heavy hitter in different attributes are recorded in different sketches. Each sketch is responsible for a distinct task, e.g., DDoS detection. Apart from heavy hitter in there are also aggregate mice flows whose frequencies are less than the threshold, which can be found at the associated filter. Since these tasks usually focus on heavy hitters, the filter only sends HH-1 flows to the corresponding sketch for further measurement.

In order to promise effectiveness and robustness of a measurement framework, Sketchtree achieves high measurement accuracy, resource efficiency and adaptivity to diversity of tasks.

**Accuracy guarantee**: these \(t + 1\) types of heavy hitters in \(t + 1\) attributes, which are recorded independently. That is, different types of flows will not collide with each other. Therefore, this mutual-independent measurement method guarantees that hash collisions happen with a very small probability. For example, after filtering in Filter 0, the number of flows has been drastically reduced in Sketch 0 and flows in Sketch 0 collide with each other with much smaller probability. Thus, the measurement accuracy can be guaranteed.

**Computation efficiency**: Sketchtree is responsible for multiple tasks. As discussed in Section 2.5, a typical solution for multiple tasks is to use multiple independent sketches, each in charge of a specific task. Compared with this solution, Sketchtree can largely reduce the total computational over-
head. Specifically, Sketchtree uses multiple filters to gradually classify the dataset into multiple subsets of flows, each corresponding to a specific task. For each task, Sketchtree further measures the task-related subset of flows, while the traditional solution needs to process all the packets in the dataset. Thus, Sketchtree can save the total computational consumption.

Adaptivity protect: these attributes are task-oriented. Due to the diversity of $t$ security-associated tasks, there are $t$ different attributes. It is worth noting that we can change the orders of these attributes by changing the index of respective filters and sketches. This enables Sketchtree to satisfy the needs of tasks with different priorities. Moreover, network measure may serve different tasks under different network circumstances. Fortunately, Sketchtree can add or delete attributes to adapt to the changes of tasks. Thus, Sketchtree is adaptive to diverse types of tasks.

4 Our Solution

4.1 Algorithm Description

We give the explanations of some parameters for ease of description.

- $t$: the number of attributes in Sketchtree.
- $T_k$: the threshold in filter $k$. For example, $T_0$ represents the threshold in Filter 0.
- $value_k(f)$: the estimated frequency of flow $f$ in Filter $k$, with $0 \leq k \leq t$. For example, $value_0(f)$ means the estimated frequency of flow $f$ in Filter 0.
- $F_{k,f}$: the aggregate mice flow that flow $f$ belongs to under attribute $k$.

**Algorithm 1** Update process algorithm for Sketchtree

1: **Input:** the incoming packet/flow $f$
2: Record flow $f$ in Filter 0
3: if $value_0(f) > T_0$ then
4: Send $f$ to Sketch 0
5: else
6: Update process for aggregate mice flows
7: $k = 1$ /*$k$ is the attribute index*/
8: while $k \leq t$ do
9: Aggregate mice flow $F_{k,f}$ is recorded by Filter $k$
10: if $value_k(F_{k,f}) > T_k$ then
11: Send flow $f$ to Sketch $k$
12: Sketch $k$ records the frequency of aggregate mice flow $F_{k,f}$
13: Break
14: else
15: if $k = t$ then
16: Send flow $f$ to Sketch $t+1$
17: Sketch $t+1$ records the frequency of flow $f$
18: end if
19: $k = k + 1$
20: end if
21: end while
22: end if

Update process of Sketchtree: since Sketchtree records the frequency of a flow on the packet level, we consider a packet in the flow. Without loss of generality, we denote the packet in the flow by $f$ as well. At first, the algorithm determines whether the flow is an elephant flow or a mice flow in Filter 0. If the estimated frequency of flow $f$ is larger than a threshold, i.e., $value_0(f) > T_0$, this flow will be considered as an elephant flow and sent to Sketch 0. Otherwise, flow $f$ is a mice flow and sent to Filter 1, which records the frequency of each aggregate mice flow in attribute 1. If an aggregate mice flow is larger than the pre-defined threshold $T_1$, it will be considered as as a heavy hitter in attribute 1 and sent to Sketch 1. Otherwise, it will be sent to the next filter. Each following filter works like Filter 1 except the last filter, namely, Filter $t$. If the aggregate mice flow in Filter $t$ is considered as small, flows in the aggregate mice flow will be sent to Sketch $t+1$. The algorithm is formally described in Alg. 1.

4.2 Filter Design and Sketch Selection

A strawman solution for filter design: one natural solution is to use a common sketch, e.g., CM or CM-CU, as a filter. In particular, this sketch is used to record the frequency of each flow. For each incoming packet in this flow, we first query the estimated frequency. If the frequency is larger than the predefined threshold, this flow will be regarded as an elephant flow, otherwise, a mice flow. However, this method faces with two challenges. (1) Memory inefficiency. The size of each counter must be able to accommodate the threshold. For example, if the threshold is 500, the size of the counter may be 16 bits. Unfortunately, most of the flows are mice flows, which means most of the counters are not fully utilized, i.e., most of counters only use a few bits to record mice flows [11]. (2) Computation inefficiency. All the hash functions are used for recording each packet. If the minimum value of the first one (or several) hashed counter(s) is less than the threshold, we can know the flow is a mice flow and the remaining hash operations can be saved. Thus, this strawman solution cannot achieve resource efficiency.

Our filter design: we use two rows of counters, namely, Layer 1 and Layer 2, each consisting of $w_1$ and $w_2$ counters respectively. The sizes of each counter at Layers 1 and 2 are different like [11], denoted by $\delta_1$ and $\delta_2$, respectively. For Layers 1 and 2 at Filter $i$, we allocate thresholds $T'$ and $T''$, with $T_i = T' + T''$, respectively. Note that Layer $i$ ($i = 1, 2$) hashes each packet into $h_i$ counters, and $v_i$ ($i = 1, 2$) means the minimum value for $h_i$ hashed counters in Layer $i$. For each incoming packet of a flow, (1) if $v_1 < T'$, the filter increments the hashed counter(s) with the minimum value by 1 (the minimum value may be located at more than one counters), and the estimated frequency of the flow is $v_1 + 1$; (2) if $v_1 \geq T'$, the filter records the flow in Layer 2. If $v_2 < T''$, the filter increments the hashed counter(s) with the minimum value by 1 in Layer 2, and the estimated value is $T' + v_2 + 1$. Otherwise, the flow overflows and will be regarded as an HH-i $1 \leq i \leq t$ flow.

Sketch selection: each sketch records heavy hitter in different attributes. After filtering, the number of measured flows is largely reduced compared with the original dataset. Thus, estimating these flows with high accuracy is accessible. To further explore the high throughput of the sketch, we would like to select the sketch with light computational overhead, e.g., CM, CM-CU, etc. Here, we choose CM-CU to measure filtered flows for its higher accuracy than Count-Min under the same parameter setting. That is, Sketch $i_{0 \leq i \leq t+1}$ is
a CM-CU sketch. Detailed parameter settings for Sketch $t_{0 \leq i \leq t+1}$ are shown in Section 6.2.

### 4.3 Two Optimization Methods for Resource Efficiency

To reduce the resource (e.g., computation and memory) consumption on switches, we propose two optimization methods to make Sketchtree work more efficiently.

#### 4.3.1 Organizing Sketchtree as a Huffman Tree (Optimization 1)

As shown in Fig. 3, Sketchtree can be recognized as a binary tree, where Filter 0 is the root of the tree. In particular, each filter is an internal node and each sketch is a leaf node. Take Filter 1 as an example. If the estimated frequency of a flow is larger than the threshold in filter 1, it will be sent to the left child node, i.e., Sketch 1, otherwise, to the right child node, i.e., Filter 2. Therefore, the processing pipeline is similar to the search operation in a binary tree. To reduce the computation consumption, we hope that the expected computational overhead for a packet should be minimized. That is, among all the sketches, we want the sketch that is similar to the search operation in a binary tree. To reduce the computation overhead, we propose two optimization methods to make Sketchtree work more efficiently. Let’s consider the first left leaf node, namely Sketch 0. It processes all the elephant flows. Since the number of packets from the elephant flows is the majority of all the packets, the placement of Sketch 0 meets our expectation to minimize the total computational overhead.

Let’s consider the placement of Sketches 1 to $t$. Each internal node (or filter) records the aggregate mice flows with a distinct attribute, which associates with a specific task. According to the architecture of Sketchtree, Filters 1 to $t$ process aggregate mice flows with attributes 1 to $t$ and the orders of these attributes can be exchanged. As we have explained in Section 3, the heavy hitters with different attributes usually do not share the same mouse flows, thus different orders of these attributes will not affect the measurement results. However, since the numbers of heavy hitters flows with different attributes are usually distinct, the orders of these attributes can influence the expected computational overhead for a packet in Sketchtree. To minimize the expected computational overhead of Sketchtree, we can exchange the placement orders of these attributes. As a result, the whole architecture becomes a Huffman tree. This adjustment can be conducted by the long-term flow management or during the system running to achieve the minimum expected computational overhead. Moreover, the stableness of Sketchtree can be maintained as we only need to exchange the indexes of filters and sketches.

#### 4.3.2 Saving Memory and Computation Usage (Optimization 2)

**Memory-saving in the filtering stage:** each filter needs to distinguish elephant flows from the data stream rather than to estimate the frequency of elephant flows. In particular, if the value of a counter exceeds the predefined threshold $T_i$, the flow will be regarded as an elephant flow and the filter does not need to update the values of hashed counters. Thus the size of a counter can be $\delta$, with $2^\delta - 1 = T_i$. For example, for the first layer in our filters, $T_1 = 15$ and we can only allocate 4 bits for each counter in Layer 1, instead of 32 bits, which is the normal size of a counter for common sketches, e.g., Count-Min [11]. As a result, $\frac{7}{8}$ of the memory size for Layer 1 is saved.

**Computation-saving:** the previous works and our experiments show that most computational overhead lies in hash operations. Thus we expect to reduce hash computational overhead and our optimization method can locate $h$ counters by one hash function. Specifically, we split the large-bit hash value (usually 32 bits [11]) into multiple segments, and each segment (or the combination of segments) is used to locate a counter. For example, for Layer 2 in the filter with $w = 2^{16}$ 16-bit counters and $h = 2$ (memory usage is 0.13 MB), we split the 32-bit hash value into two 16-bit segments to locate two counters, respectively. As an extreme case, if the 32-bit hash value is not big enough to be divided into $h$ independent $\delta$-bit hash values, we will apply combinations of segments to locate the counters. For example, for Layer 1 in the filter with $w = 2^{18}$, $\delta = 4$ and $h = 3$ (memory usage is 0.13 MB), we separate the 32-bit hash value into four segments, one 11-bit segment and three 7-bit segments. Each 7-bit segment is combined with 11-bit segments to be an 18-bit value, which can be used to locate a counter. Note that although the extreme case is used by some literatures [11], the sharing of bits may affect the independence among the among segmented hash values to a certain extent.

### 5 Performance Analysis

We first analyze the computational cost of Sketchtree by comparing the hash times per packet. Let $d_1$ and $d_2$ be the numbers of hash operations per packet for a filter and a sketch, respectively. The portion of packets processed in Sketch $i$ (0 $\leq i \leq t + 1$) is $\beta_i$ with $\sum_i \beta_i = 1$ and accordingly, the portion of packets traversing in Filter $i$ is $\sum_{j=0}^{t+i} \beta_j$. Averagely, the hash times per packet is $d_1 (\sum_{j=0}^{t+i} \beta_j) 1 + d_2 = d_1 (\sum_{j=0}^{t+i} \beta_j) + d_2$. Recall that the hash times per packet of the second baseline solution is $(d_1 + \beta \cdot d_2) \cdot (t + 1)$. By redundant scaling, the hash times per packet of Sketchtree is:

$$d_1 (\sum_{i=0}^{t} \beta_i \cdot i) + d_2 = d_2 + d_1 \cdot (1 + \sum_{i=1}^{t} \beta_i \cdot i) \leq d_1 + d_2 + d_1 (1 - \beta_0) \cdot t \cdot (t + 1)$$

The number of packets recorded in Sketch 0 is the majority of all the packets in the data stream, i.e., $\beta_0 \rightarrow 1$. Practically, we can have $\beta_0 = 0.8$ [51] [52]. Therefore, the hash times per packet of Sketchtree is about $d_1 + d_2 + 0.2d_1 \cdot t$ and the ratio to that of the second baseline solution is:

$$\frac{d_1 + d_2 + 0.2d_1 \cdot t}{(d_1 + 0.8d_2) \cdot (t + 1)} \approx \frac{\frac{0.2(d_1 \cdot t + d_2)}{(d_1 + 0.8d_2) \cdot (t + 1)}}{t + 1}$$

For simplicity, we assume $d_1 = d_2$ and Eq. (2) becomes:

$$\frac{1}{t + 1} + \frac{0.2(d_1 \cdot t + d_2)}{(d_1 + 0.8d_2) \cdot (t + 1)} \approx \frac{1}{t + 1} + \frac{0.2}{1.8} = \frac{1}{t + 1} + \frac{1}{9}$$

Based on the above analysis and assumptions, we give the following theorem.

**Theorem 1.** The hash times per packet of Sketchtree is approximately $\frac{1}{t + 1} + \frac{1}{9}$ times of that of the second baseline solution, where $t + 1$ is the number attributes that need to be measured.
Therefore, Sketchtree can largely reduce the computing cost compared with the second baseline solution, let alone the first baseline solution which requires more hash times per packet.

In the following, we will analyze the accuracy of Sketchtree. As mentioned in Section 4.2, Sketchtree selects a previous sketch, i.e., CM-CU, for traffic measurement. Note that CM-CU is an improved sketch of Count-Min, and its measurement error is also bounded by the analysis of Count-Min [10]. Here, we do not repeat the analysis of CM-CU and this section mainly focuses on the performance of filters, i.e., the misreport rate of each filter. Without loss of generality, we consider an arbitrary filter whose threshold is \( T \) with \( T \in \{T_0, T_1, \ldots, T_n\} \) and a time window \([1, E]\) from time point 1 to time point \( E \). In this time window, there are \( n \) flows in the dataset, denoted as \( \Gamma = \{f_1, f_2, \ldots, f_n\} \). Flow \( f \in \Gamma \) includes \( N_f \) packets. Before performance analysis of Sketchtree, we first give the definition about frequency distribution of data stream \( \Gamma \).

**Definition 4.** For an arbitrary time point \( j \leq j \leq E \), \( I_k[j] \) is the subset of flows whose current frequencies are not less than \( k \). Formally, \( I_k[j] = \{ f | N_f[j] \geq k, k \in Z^+ \} \). Obviously, \( I_k[j] = I_k[j] - I_{k+1}[j] \) means the subset of flows whose current frequencies are exactly \( k \).

According to Alg. 1, flows whose frequencies are larger than \( T \), will be identified as a heavy hitter, and sent to the corresponding sketch. Thus, Sketchtree only misreports some mice flows whose real frequencies are smaller than \( T \) as each filter may produce false-positive error. For simplicity, we say these mice flows are misreported and denoted by \( I_{mr} \). Thus, the misreport rate \( P_{mr} \) is defined as follows:

\[
P_{mr} = \frac{I_{mr}}{I[E] - I_T[E]} \tag{4}
\]

We first adopt the theory of standard Bloom filter to derive the \( P_{mr} \) of CM-CU. A standard Bloom filter [53] consists of \( w \)-bit array associated with \( d \) hash functions. When a packet arrives, each hash function in the filter hashes the packet to a counter, and sets the counter to one. When querying a flow, if all the \( d \) hashed counters are one, it reports true; otherwise, false. Formally, given \( w \), \( d \) and the number of processed flows \( n \), the false positive rate \( P_{fp} \) in the standard Bloom filter [53] is expressed as

\[
P_{fp}(w, d, n) = \left(1 - \left(1 - 1/w\right)^{nd}\right)^d \approx \left(1 - e^{-\frac{nd}{w}}\right)^d \tag{5}
\]

Here we have the following lemma for function \( P_{fp}(w, d, x) \):

**Lemma 2.** Function \( P_{fp}(w, d, x) = \frac{1}{x} \sum_{i=0}^{x-1} P_{fp}(w, d, i) \), with \( \forall x \in Z^+ \), is a monotonic increasing function of \( x \).

**Proof:** If \( P_{fp}(w, d, n) \) in Eq. (5) is a monotonic increasing function of the variable \( n \). For simplicity, \( P_{fp}(w, d, i) \) is denoted by sequence \( \{a_i\} \). Thus, we have \( a_i < a_{i+1} \). For \( \forall k \in Z^+ \), we have:

\[
\begin{align*}
 a_i < a_{i+1} & \iff \sum_{i=0}^{k-1} a_i < k \cdot a_k \\
 & \iff \sum_{i=0}^{k-1} a_i < \sum_{i=0}^{k-1} a_i < k \cdot a_k + \sum_{i=0}^{k-1} a_i \\
 & \iff (k + 1) \sum_{i=0}^{k-1} a_i < k \sum_{i=0}^{k-1} a_i \iff \frac{1}{k+1} \sum_{i=0}^{k-1} a_i < \frac{1}{k} \sum_{i=0}^{k-1} a_i \\
 & \iff P_{fp}(w, d, k) < P_{fp}(w, d, k + 1)
\end{align*}
\]

The lemma holds. \( \square \)

**Lemma 3.** The subsets of flows, \( I_k[j] \) and \( J_k[j] \), have the following relation:

\[
\begin{align*}
|I_k[j]| & \leq |J_k[j]| \\
& \leq |I_k[j]| + \sum_{i=1}^{k-1} \left[ (|I_k[j]| - |I_{k+i}[j]|) \cdot \prod_{u=1}^{i} P_{fp}(w, d, |J_{k-u}[j]|) \right] \tag{7}
\end{align*}
\]

where \( P_{fp}(w, d, x) = \frac{1}{x} \sum_{i=0}^{x-1} P_{fp}(w, d, i), \forall x \in Z^+ \).

**Proof:** We introduce a multi-layer Bloom filter to build the relation between standard Bloom filter and CM-CU. This multi-layer Bloom filter is an array of standard Bloom filters with the same \( w \), \( d \) and hash functions. Each Bloom filter is associated with level. For each incoming packet, the multi-layer Bloom filter will check levels from 1 to \( k \). It stops when level 1, 1 \( \leq i \leq k \), reports false, and we will set the \( d \) hashed counters to one (also called true). After processing all the \( x \) distinct flows in the Bloom filter, how many flows are expected to be reported true by mistake? The answer is \( P_{fp}(w, d, x) \cdot x \). Specifically, we consider the false positive rate of each flow. For the \( i \)th incoming flow, its false positive rate equals to \( P_{fp}(w, d, i - 1) \), as there are total \( i - 1 \) flows inserted to the Bloom filter previously. Thus, the expected number of distinct flows that are reported true by mistake is \( \sum_{i=0}^{x-1} P_{fp}(w, d, i) = P_{fp}(w, d, x) \cdot x \).

Due to the false positive rate, the estimated frequency of a flow may be larger than its real frequency. Similar to [11], the contribution to \( J_k[j] \) derives from the following \( k \) parts: (1) flows in set \( I_k[j] \); (2) flows in set \( I_{k-1}[j] - I_k[j] \) and experiencing one or more false positives from level 1 to level \( k - 1 \); (3) flows in set \( I_{k-2}[j] - I_{k-1}[j] \) and experiencing two or more false positives from level 1 to level \( k - 1 \);...; (k) flows in set \( I_{k-2}[j] - I_{k-1}[j] \) and experiencing \( k - 1 \) false positives from level 1 to level \( k - 1 \). For the first part, the contribution to \( J_k[j] \) is \( I_k[j] \); for the second part, since there are at most \( k - 1 \) flows in level \( k - 1 \), the maximum probability of experiencing false positives is \( P_{fp}(w, d, J_{k-1}[j]) \); for the third part, the size of set is \( |I_{k-2}[j] - I_{k-1}[j]| \), and the maximum probability of experiencing such false positives is \( P_{fp}(w, d, J_{k-2}[j]) \cdot P_{fp}(w, d, J_{k-2}[j]) \),... for the \( k \)th part, the size of set is \( |I_{k-1}[j] - I_{k-2}[j]| \), and the maximum probability of experiencing such false positives is \( \prod_{i=1}^{k-1} P_{fp}(w, d, J_{k-i}[j]) \). Considering all the above \( k \) parts, the lemma holds. \( \square \)

As \( P_{fp}(w, d, x) \) is a monotonic function of \( x \), \( |J_k[j]| \) can be bounded recursively by Lemma 2. For simplicity, \( |J_k[j]| \) and \( |J_k[j]| \) represent the lower and upper bounds of \( |J_k[j]| \), respectively. We first give a general bound of \( P_{mr} \) in Sketchtree.
Theorem 4. The misreport rate of Sketchtree satisfies

\[ P_{mr} \leq \frac{\sum_{k=1}^{T} \left( 1 - \prod_{u=1}^{k} \left( 1 - P_{ fp}(w, d, t_i[t_u]) \right) \right) \cdot |\Delta_k[E]|}{\sum_{k=1}^{T} |\Delta_k[E]|} \]

where \( t_u \), with \( 1 \leq u \leq k \), represents the incoming time point of the \( u \)th packet in the flow that includes \( k \) packets. Obviously, \( t_u \) depends on the flow/packet distribution of appearance time.

Proof: For each misreported flow, its estimated frequency must exceed the threshold \( T \), thus we divide the \( P_{mr} \) into \( T \) parts: \( P_{mr}^1, P_{mr}^2, ..., P_{mr}^T \), where \( P_{mr}^k \) denotes the misreported rate of flows whose real frequencies are \( k \), namely \( \Delta_k[E] = \{ f | N_f = k \} \). Obviously, we have:

\[ P_{mr} = \frac{\sum_{k=1}^{T} (P_{mr}^k \cdot |\Delta_k[E]|)}{\sum_{k=1}^{T} |\Delta_k[E]|} \tag{9} \]

Here we consider an arbitrary flow \( \gamma \in \Delta_k[E] \), which appears in the time window \( [1, E] \) \( k \) times. Note that \( t_1, t_2, ..., t_k \) are the \( k \) appearance time points. If flow \( \gamma \) is misreported and its real frequency is \( k \), the flow must experience \( T - k + 1 \) false positives from level 1 to level \( T \). Note that if \( P_{mr}^k \leq 1 \leq k \leq T \), the rate is that misreport happens exactly in the time \( t_u \), and we derive:

\[ P_{mr}^k = 1 - \prod_{u=1}^{k} (1 - P_{ fp}(w, d, t_i[t_u]) \tag{10} \]

For each \( P_{mr}^k \), since the misreport happens in time point \( t_u \), the \( T - k + 1 \) times of false positives happen before or equal to \( t_u \). That is, \( t_u \) is the maximum time point among these appearance time points of \( T - k + 1 \) false positives. Since the \( P_{ fp}(w, d, x) \) is the monotonic increasing function of \( x \), we have:

\[ P_{mr}^k \leq \prod_{u=1}^{T-k+1} P_{ fp}(w, d, t_i[t_u]) \tag{11} \]

By putting Eqs. (10) and (11) into (9), the theorem holds. \( \square \)

6 Performance Evaluation

6.1 Experimental Setup

We have implemented Sketchtree, Cold Filter [11], CM [10] and CM-CU [25] sketches, and compared their performance through experiments under real traffic dataset from CAIDA [55] in 2013. The source code has been publicized at Github [46]. We have two scales of the CAIDA dataset. The first scale of the CAIDA dataset lasts for 1min and contains 18M packets and 0.43M flows in which each flow is identified by its source and destination addresses. In order to create a larger dataset for our evaluation, we have also combined three consecutive 1-minute CAIDA datasets as the second scale of dataset. This combined dataset lasts for 3min and contains 54M packets and 1.03M flows. As for the measurement interests, we adopt two security-associated tasks, i.e., detection of network scan and DDoS. To serve these two tasks, Sketchtree cares about heavy hitters in attributes of source and destination addresses respectively, i.e., \( t=2 \). According to Optimization 1 in Section 4.3.1, the order of filters processing flows in different attributes can be adjusted to minimize the computational overhead of Sketchtree. At the beginning of the system running, since flow distribution on each sketch of Sketchtree is unknown, we arbitrarily put the source address as the first attribute before the destination address. The impact of Optimization 1 on the computational overhead of Sketchtree will also be studied.

6.2 Benchmarks and Parameter Settings

We have introduced two kinds of baseline solutions in Section 2.5. Accordingly, We adopt two categories of benchmarks. The first category of benchmarks, including Count-Min (CM) and CM-CU, stores the information of all the data stream in one independent data structure for each attribute. The second category of benchmarks, i.e., CF+CM-CU, applies filters to coarsely measure the information of flows and only records flows whose estimated value is above a threshold in the second CM-CU sketch stage. Note that these two categories of benchmarks measure data stream in different attributes in independent data structures. Here, we have three attributes, which are based on source-destination address, source address, and destination address. Accordingly, these two categories of benchmarks will apply three independent data structures for data flow measurement. The detailed description of Sketchtree and these benchmarks is as follows. Aside from these two categories of benchmark, we have also introduced a natural method in Section 1, in which the size of aggregate mice flow can be inferred by measuring individual mice flows and then adding up them.
together. The results shown in Figs. 5 and 6 will be explained in detail later. But we find that this method is not efficient and practical enough. Therefore, we do not consider this method as a comparable benchmark for further comparison.

**Sketchtree**: according to the experimental setup, there are three attributes and therefore Sketchtree will process the data stream with three filters and three sketches. Let $M_s$ be the total memory size of Sketches 0 to 3, and $M_f$ be the total memory size of three filters. We set $M_s : M_f = 1 : 1$, which is similar to [11]. Each filter is implemented by two layers to ensure measurement effectiveness, and each sketch is equipped with $d$ arrays, each associated with $w$ counters. The values of $d$ and $w$ refer to [11] and [20] and are suitable for the scale of our dataset. Note that the size of each counter is $d$ bits, which is determined by our optimization method (see Section 4.3.2). $M$ is the memory size of the layer and $h$ is the number of hash functions in each layer of filters. For ease of description, $T$ represents the threshold of an arbitrary layer in the filter and its value setting is consulted from [11].

### Table 2: Parameter Settings for Each Sketch/Filter

<table>
<thead>
<tr>
<th>Sketch/Filter</th>
<th>Memory Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM</td>
<td></td>
</tr>
<tr>
<td>CM-CU</td>
<td></td>
</tr>
<tr>
<td>Sketchtree</td>
<td></td>
</tr>
</tbody>
</table>

#### CM: there are three independent CM sketches to record data stream in attributes of source-destination address, source address, and destination address, respectively. To explore high measurement accuracy and conform the common setting in CM [10], each CM sketch is allocated 4 rows of counters and each row is equipped with a hash function. The size of each counter is 32 bits [11].

#### CM-CU: three independent CM-CU sketches are employed. Since CM-CU is an improved CM sketch, parameter settings of CM-CU are the same as those of the above CM sketch.

**CF+CM-CU**: it contains a filter named CF (Cold Filter [11]) and a CM-CU sketch. Specifically, CF separates elephant flows and mice flows, and only sends elephant flows to CM-CU for further measurement. CF+CM-CU is applied to record individual flows, especially elephant flows. Specifically, CF includes two layers with the same parameter settings as Filter 1 in Sketchtree for fairness. CM-CU contains 3 rows of counters each with a hash function. The size of each counter is 32 bits. As recommended in [11], the memory of CF is equal to that of CM-CU. For each attribute, a CF+CM-CU sketch is assigned and there are totally three CF+CM-CU data structures.

### 6.3 Metrics

**Average absolute error (AAE) in frequency estimation**: AAE is formulated as $\frac{1}{|\Gamma|} \sum_{f \in \Gamma} |\hat{N}_f - N_f|$, where $\hat{N}_f$ is the estimated frequency of flow $f$, $N_f$ is the real frequency of flow $f$, and $\Gamma$ is the flow set. We calculate AAE by querying the absolute error of each flow.

#### Inserting speed and CPU cycles per packet (Figs. 7-8): we evaluate the computational overhead for Sketchtree and all other benchmarks with two different scales of datasets. In Figs. 7 and 8, Sketchtree achieves the fastest inserting speed, compared with other three benchmarks. Specifically, the inserting speed of Sketchtree is two times of that of CF+CM-CU. Compared with CM and CM-CU, Sketchtree improves the inserting speed by 101% and 123%, respectively. Comparing filter based solutions like CF+CM-CU with CM and CM-CU, we can see that filter based solutions can increase the inserting speed. However, for different attributes, CF+CM-CU still processes data stream with independent data structures, which will cause repeated processing procedures. That’s the reason for the 100% inserting speed improvement of Sketchtree compared with CF+CM-CU.
AAE for heavy hitters in different attributes that exceed the traffic size of 1000 (Figs. 4 and 9): In Section 2.4, heavy hitters (HH) is large flows whose traffic size exceeds a threshold. We first set the threshold as 1000 and AAE of HH in three attributes is presented in Figs. 4-9. The memory size ranges from 1MB to 10MB. As we can see, the filter based solutions like CF+CM-CU can improve the measurement accuracy compared with classical CM and CM-CU. Furthermore, our Sketchtree solution based on hierarchical filtering can reduce the estimated error even compared with CF+CM-CU for HH in attribute of source or destination address, and achieves the same AAE for HH in attribute of src-dst address as CF+CM-CU (shown in Figs. 4(a), 9(a), 10(a) and 11(a)). Specifically, by Figs. 4(b)-4(c), Sketchtree can reduce AAE of HH in attributes of source address and destination address by 30% and 50%, respectively, compared with CF+CM-CU. The reason is that HHs in attribute source or destination address under Sketchtree do not collide with src-dst-attribute elephant flows like they would in the respective CF+CM-CU filer and sketch. Moreover, under a larger data stream like the 3-minute dataset, Sketchtree can still maintain measurement accuracy. For instance, even in a compact memory size of 1MB, Sketchtree can achieve less than 25 of AAE of HH, while other benchmarks like CM can be up to 250. Therefore, Sketchtree is efficient for the measurement of HH with a threshold of 1000.

AAE for heavy hitters in different attributes that exceed 5000 of traffic size (Figs. 10 and 11): We have also investigated the performance of Sketchtree and benchmarks when the threshold for HH is set as 5000. Under this circumstance, the number of HHs is reduced compared with the definition of HH above where the threshold is 1000. The memory size also ranges from 1MB to 10MB and AAE of HH in three different attributes is presented. As we can see, the measurement accuracy of Sketchtree is improved. For example, when the memory size is 1MB, AAE of HH in the attribute of source-destination address (shown in Fig. 10(a)) is less than 10. Compared with AAE of HH with the threshold of 1000 in Fig. 4(a), here Sketchtree reduces AAE by 20%. However, under the same memory size, i.e., 1MB, other solutions like CM can only achieve AAE is 267 for HH in attribute of source-destination address by Fig. 11(a), and its AAE is reduced to 20 only when its memory size exceeds 10MB. By comparison, we can see that Sketchtree achieves measurement accuracy with less memory consumption.

Summaries: (1) Thanks to the one “big” sketch concept,
Impact of threshold setting of filters on inserting speed

6.5 Sensitivity Analysis

Impact of threshold setting of filters on inserting speed (Fig. 12): according to the architecture of Sketchtree in Fig. 3, flows whose estimated value is less than the threshold of filters will be regarded as mice flows and sent to the layer of filter for measurement in the next attribute, while those above the threshold will be sent the corresponding sketch for more precise measurement. Therefore, we observe the impacts of the threshold of filters on the inserting speed and the results are shown in Fig. 12. The inserting speed decreases only a little and is not sensitive to the threshold. For instance, the inserting speed decreases by 0.2Mpps when the threshold increases from 250 to 1250. There may be two reasons. (1) The majority of packets/traffic in the data stream is from the elephant flows whose size is usually over $10^4$ or even $10^5$, and these elephant flows will not be affected by the threshold settings. (2) For some medium size flows, they may be sent to the next attribute for further recording when the threshold of filters increases over their traffic sizes, which will increase the processing overhead. Fortunately, even when the threshold increases to 1250, Sketchtree can still preserve the significant improvement of inserting speed compared with all other benchmarks.

Impact of Optimization 1 on inserting speed and CPU cycles per packet (Fig. 13): in this section, we observe the impacts of Optimization 1 on different metrics. That is, Filters 1 and 2 measure aggregate mice flows in attributes of destination address and source address, respectively. For ease of reading, Sketchtree with Optimization 1 is denoted Sketchtree+Opt1 in the following figures. We present the HH distribution on different sketches in Sketchtree and the results are shown in Fig. 13(a). There are about 340 and 450 heavy hitters in Sketch 1 and Sketch 2 of Sketchtree. That means, more heavy hitters are processed in Sketch 2, which consumes more processing overhead as it is far from the root of Sketchtree. Therefore, in our optimization, we exchange the order of these two attributes and compare the inserting speed of Sketchtree+Opt1. The results in Fig. 13(b) show that Optimization 1 can improve the inserting speed by about 0.4Mpps. It is worth noting that the larger the gap of the number of HH between different attributes is, the more significant improvement of inserting speed becomes.

Summaries: (1) The inserting speed of Sketchtree is not very sensitive to the threshold of filters and Sketchtree can preserve the advantages of fast inserting speed over other sketches. (2) Our Optimization 2 can reduce the inserting speed and show more potential if the number of HHs in different attributes varies a lot.

6.6 A Study Case on Network Scan and DDoS Dataset

We evaluate the measurement accuracy of all algorithms on a real dataset. The dataset includes a DDoS attack and a small-scale network scan from Lincoln Laboratory [56]. The memory space is 1 MB. We present two kinds of results. The first is AAE and the second is the measurement accuracy of the two attacks. The results are shown in Table 3 and Fig. 14. From Fig. 14, we can see that Sketchtree and CF+CM-CU achieve the most accurate measurement compared with CM and CM-CU for attribute of src-dst address, which is consistent with the result under previous datasets in Fig. 9. As for the measurement accuracy for the network scan and
DDoS attack in Table 3, Sketchtree and CF+CM-CU achieve the precise measurement while CM and CM-CU produce the measurement result with errors. The measurement result of Sketchtree validates our assumption that the security-associated mice flows are usually disjoint.

7 Conclusion

In this paper, we propose an accurate traffic measurement framework named Sketchtree to record heavy hitters in different attributes. In particular, Sketchtree inherits the idea of filtering and develops it into the multi-attribute filtering. For different attributes, Sketchtree will pre-select the heavy hitters by filtering and send them to the respective sketch for further measurement. All the processing procedure goes in one "big" sketch. Compared with the traditional method where each task is fulfilled by one specific sketch, our solution can largely reduce resource consumption. We also present some optimization methods to further improve the resource efficiency of Sketchtree. Experimental results show that Sketchtree can largely reduce measurement errors of heavy hitters in different attributes and improve the processing speed compared with state-of-the-art solutions under the same memory consumption.

8 Acknowledgment

This research is supported by the National Science Foundation of China (NSFC) under Grants 61822210, 61936015, and U1709217; and by Anhui Initiative in Quantum Information Technologies under No.AHY150300.

References


Liusheng Huang received the M.S. degree in computer science from University of Science and Technology of China in 1988. He is currently a senior professor and Ph.D. supervisor of the School of Computer Science and Technology at the University of Science and Technology of China. He has published 6 books and more than 300 journal/conference papers. His research interests are in the areas of internet of thing, vehicular ad hoc network, information security, and distributed computing.

Yutong Zhai received the B.S. degree in computer science from University of Science and Technology of China in 2017. He is currently a PhD student in the School of Computer Science and Technology at the University of Science and Technology of China. His research interests are in the areas of software defined networking, Internet traffic measurement and edge computing.